




ILP and Its Exploitation

 Author	 Lee Edith.D
 Last Update	@March 31, 2022

Background

Pipeline CPI

ILP

Dependences and Hazards

3 Types of Dependences

Data Hazards

Basic Compiler Techniques

Latencies of FP operations

Loop Unrolling

Reducing Branch Costs with Advanced Prediction

Control Speculation

What if mis-speculated?

Branch Prediction

Predict What?

Categorizing Branches

Static Branch Prediction

Dynamic Branch Predictor

1. Simplest

Dynamic Scheduling

2 Algorithms for DS

Scoreboard Approach

Tomasulo's Approach

Background

- Instruction Level Parallelism (ILP)
 - overlapping execution of inst.
- 2 methods:
 - Hardware-based dynamic approaches
 - Dynamic Branch Predictor

- Dynamic Scheduling
 - scoreboard
 - algorithm
- Compiler-based static approaches
 - Static scheduling
 - Loop unrolling

Pipeline CPI

- when exploiting instruction-level parallelism, the goal is to minimize CPI
- **Pipeline CPI = Ideal pipeline CPI + Structural Stalls + Data Hazard Stalls + Control Stalls**
 - Ideal pipeline CPI = 1 (N/N)
 - goal: reduce CPI
- Efforts to avoid hazards
 - forwarding & bypassing
 - delayed branch
 - software method, ask compiler to schedule must-to-do inst.
 - basic compiler pipeline scheduling
 - software method
 - basic dynamic scheduling
 - loop unrolling
 - branch prediction
 - dynamic scheduling
 - reduced Write-After-Write (WAW)

ILP

- Basic block

- a straight-line code sequence with no branches in
- ILP in basic block is limited
 - basic block is small = 3~6 inst.
 - branch frequency is high = 15~25%
- need to optimize across branches

Dependences and Hazards

- Dependences
 - ILP is hindered by dependences in programs
 - but not all, which not cause hazards and allow for ILP
- Hazards
 - when there's hazards, the pipeline need to be stall

3 Types of Dependences

- data dependency
 - possibility of a hazard
 - eg. **lw** and **add** (R-type), hazards occurs if they're closed
 - Order
 - some inst. orders can not be rescheduled
- name dependency
 - 2 ints. use the same register or memory location but no flow of information
 - Anti-dependence
 - inst. j writes a register or memory location that inst. i reads
 - order must be preserved
 - Output-dependence
 - inst. i and inst. j write the same register or memory location
 - order must be preserved

- To resolve
 - *renaming*
 - compiler converts register names when executed
 - hardware converts register names when executed
- control dependency
 - inst. under control dependency cannot be moved before branch inst.

Data Hazards

- Read-after-Write (RAW)
-
- Write-after-Write (WAW)
- Write-after-Read (WAR)

Basic Compiler Techniques

Latencies of FP operations

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

A compiler's ability to perform pipeline scheduling is dependent both on the amount of ILP available in the program and on the latencies of the functional units in the pipeline

- Compiler:

- schedule inst. to avoid pipeline stall
- Basic compiler techniques to exploit ILP
 - Static scheduling
 - Loop unrolling

Loop Unrolling

Reducing Branch Costs with Advanced Prediction

Control Speculation

- Execute inst. beyond a branch before the branch is resolved

What if mis-speculated?

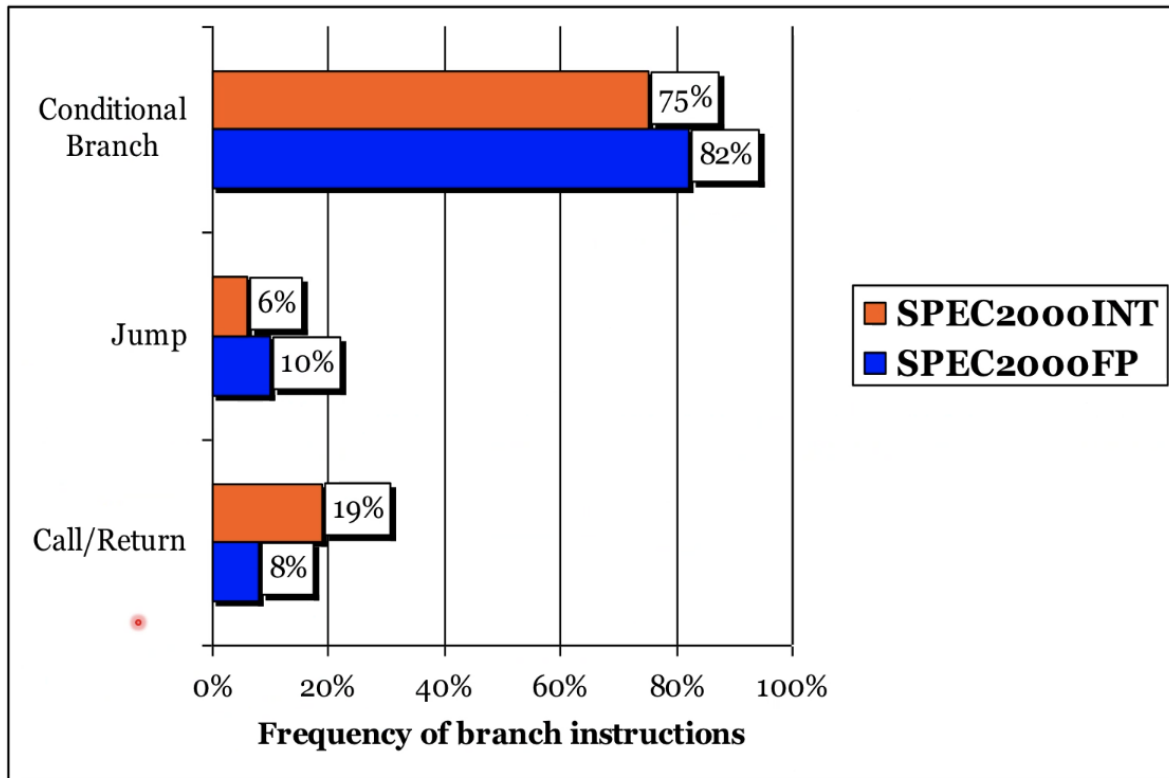
- Recovery mechanism
- Squash inst. on the incorrect path

Branch Prediction

- Dynamic
- Static

Predict What?

Categorizing Branches

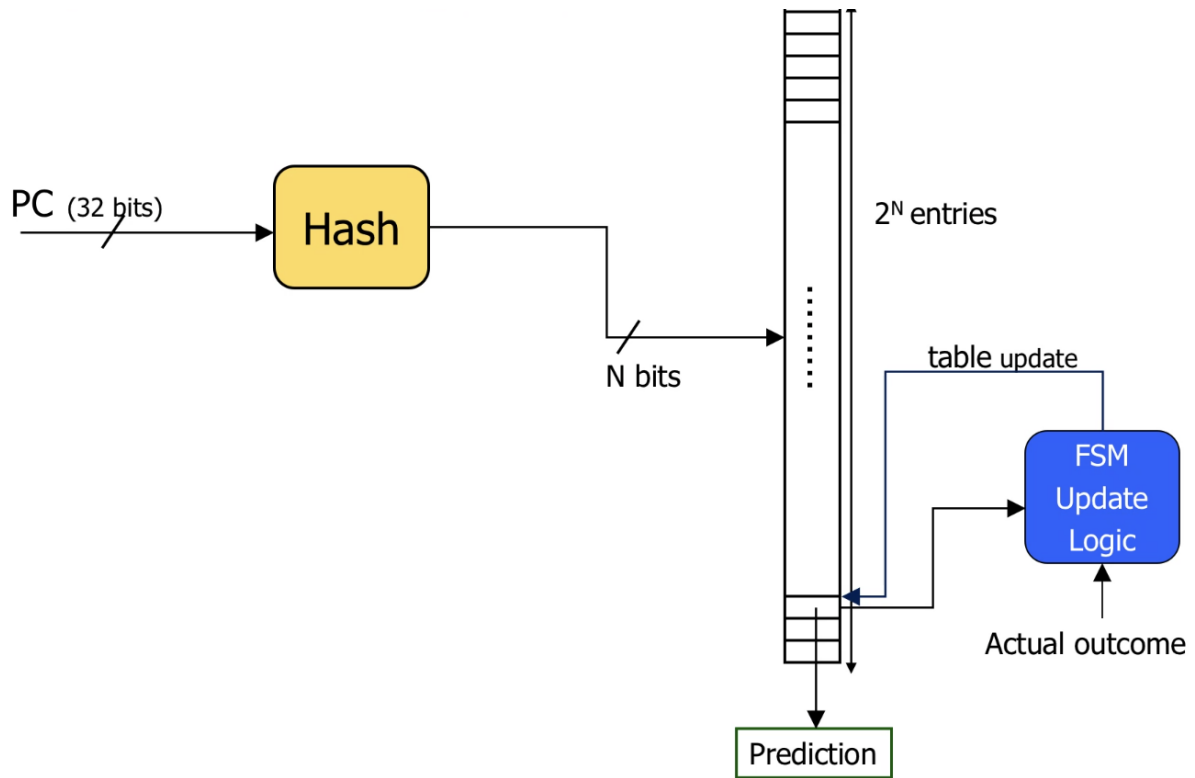


Source: H&P using Alpha

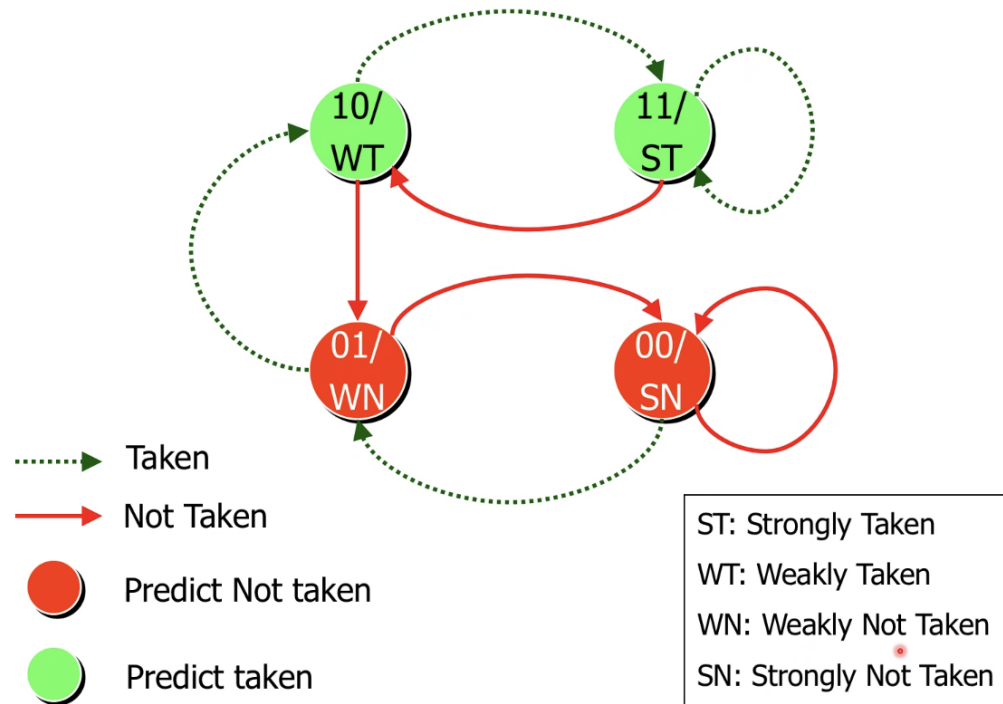
Static Branch Prediction

Dynamic Branch Predictor

1. Simplest



- 1-bit branch history table (hash table)
- based on the latest outcome
- FSM of the Simplest Predictor
 - 1-bit
 - 2-bit Saturating Up/Down Counter Predictor



Dynamic Scheduling

- rearrange order of inst. to reduce stalls while maintaining data flow
- implement by hardware
- In-order issue, but out-of-order execution
 - cons: substantial increase in hardware complexity
 - cons: L complicates exceptions
- Let an instruction begin execution as soon as its data operands are available → may cause WAR&WAW hazards

2 Algorithms for DS

Scoreboard Approach

- divide ID stage in to 2:
 - Issue
 - decode & check for structural hazards

- Read operands
 - wait until no data hazards
- In-order issue, out-of-order execution
- Inst. only execute whenever not dependent on previous inst. and no hazards
- Solutions for WAR
 - Detect hazard and **stall writeback** until registers have been read
 - Read registers only during Read Operands stage
- Solutions for WAW
 - Detect hazard and **stall issue** of new instruction until other instruction completes
- 4 stages in scoreboard algorithm
 - issue
 - only issue when there's no structural hazards & WAW
 - in0order
 - read operands
 - only read when all read flags are available
 - exe
 - out-of-exe
 - writeback
 - only write back when there's no WAR

Tomasulo's Approach

- Eliminate WAR and WAW by register renaming
 - extra register
 - reserve stack